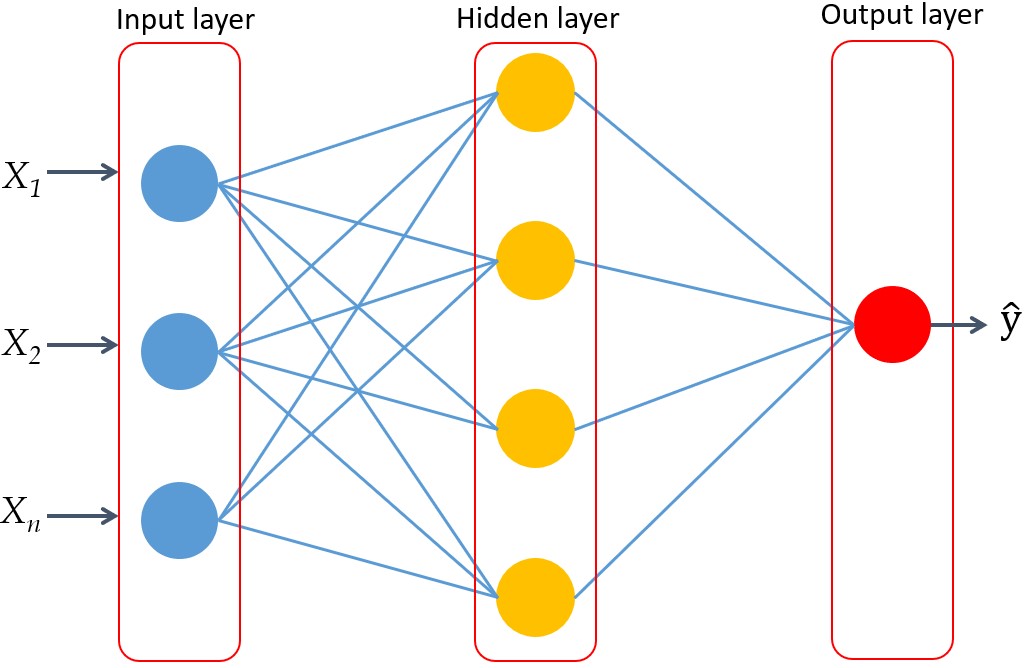
**MEMBANGUN MODEL ARTIFICIAL NEURAL NETWORK (ANN)**

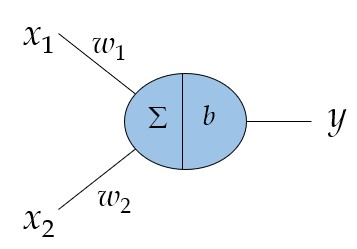
*Artificial Neural Network* (ANN) merupakan salah satu metode mesin pembelajaran yang terinspirasi oleh cara kerja jaringan saraf di otak manusia dalam menyelesaikan masalah melalui perubahan bobot sinapsisnya. ANN adalah metode *supervised learning* yang mampu mempelajari pola suatu data dengan menggunakan rekam data di masa lalu sebagai data latih. Pada ANN, terdapat neuron yang terhubung dengan neuron lainnya. Neuron-neuron tersebut akan mentransformasikan informasi yang diterima menuju ke neuron yang lain. Metode ANN merupakan arsitektur *Fully Connected Network* (FCN) dimana setiap hubungan antar neuron ini terdapat nilai bobot. Arsitektur FCN terdiri dari 3 layer, yaitu input layer, hidden layer, dan output layer. FCN dapat diklasifikasikan menjadi 2 (dua) jenis, yaitu single-layer dan multi-layer.

1. Arsitektur single-layer terdiri dari 2 layer, yaitu input dan output. Input layer menerima masukkan, sedangkan output layer memberikan respon berdasarkan masukkan.
2. Arsitektur multi-layer seperti pada Gambar 1 memiliki struktur tambahan yaitu hidden layer yang menghubungkan input layer dan output layer. Jumlah hidden layer merepresentasikan tingkat kompleksitas sebuah jaringan, semakin kompleks permasalahan maka akan dibutuhkan semakin banyak jumlah hidden unit.



## Gambar 1. Arsitektur Multi-layer FCN

Jumlah *neuron* pada *input layer* merepresentasikan jumlah fitur pada sebuah dataset, jika input berupa data citra maka jumlah neuron sesuai ukuran citra input. Jumlah *neuron* pada *hidden layer* umumnya lebih banyak daripada jumlah neuron di *output layer*. Sedangkan jumlah *neuron* pada *output layer* sesuai dengan jumlah kelas pada permasalahan klasifikasi. Pada kasus prediksi dan klasifikasi dua kelas, jumlah *neuron* pada *output layer* adalah satu neuron.

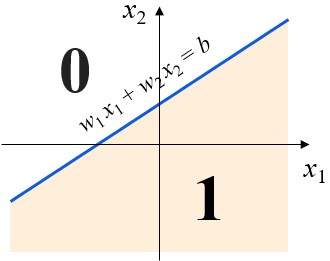


## Gambar 2. Representasi sebuah neuron pada metode ANN

Gambar 2 menunjukkan bahwa setiap *neuron* mempunyai luaran *y* yang diperoleh dari persamaan linear yaitu penjumlahan dari perkalian input *x* dan bobot *w* dan menambahkan bias *b*. Perhitungan luaran pada setiap neuron dapat dilihat pada persamaan 1.

ŷ  𝑤𝑖𝑥𝑖 + 𝑏) (1) Dimana *a* adalah fungsi aktivasi.

Sebuah neuron pada ANN dapat menyelesaikan permasalahan klasifikasi biner sebagai fungsi pemisah (*hyperspace separation*) dan *binary threshold* seperti ditunjukkan pada Gambar 3.



## Gambar 3. Sebuah neuron dapat menyelesaikan permasalahan klasifikasi biner

Permasalahan klasifikasi dapat dikategorikan sebagai:

* Permasalahan Linear, misalnya fungsi OR dan AND
* Permasalahan Non-Linear, misalnya fungsi XOR

Pada ANN, fungsi aktivasi dapat merubah neuron menjadi non-linear. Beberapa contoh fungsi aktivasi yang umum digunakan pada metode ANN sebagai berikut: 1. Fungsi Sigmoid

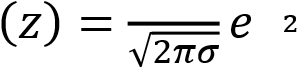


𝑎(𝑧) = 𝑧 (2)



1. Fungsi Gaussian

𝜇

𝑎 𝜎 (3)

1. Fungsi Hyperbolic Tangent

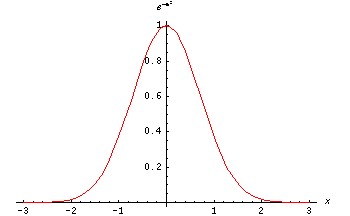
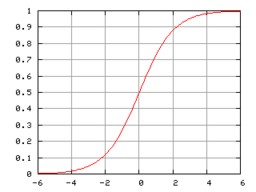
𝑒𝑧− 𝑒−𝑧

𝑎(𝑧) = 𝑒𝑧+𝑒−𝑧 (4)

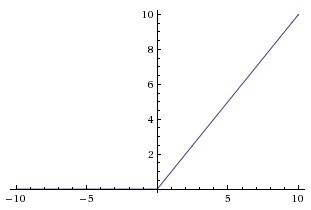
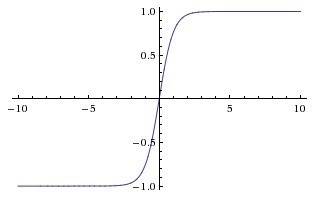
1. Fungsi Rectified Linear Unit (ReLU)

𝑎(𝑧) = max (0, 𝑧) (5)

Adapun grafik dari fungsi aktivasi dapat dilihat pada Gambar 4.



(a) Sigmoid (b) Gaussian



(c) Hyperbolic Tangent (d) ReLU

## Gambar 4. Contoh grafik fungsi aktivasi

Pada permasalahan non-linear dan permasalahan yang lebih kompleks, dapat menggunakan arsitektur Multi-layer ANN.

# Desain Arsitektur ANN

Terdiri dari tiga layer yaitu:

* input layer
* hidden layer
* output layer

Hubungan antar neuron pada ANN merupakan *Fully connected layer*. Jumlah *hidden layer* sebaiknya disesuaikan dengan kompleksitas permasalahan. Jumlah neuron pada *hidden layer* umumnya lebih banyak daripada jumlah *neuron* di *output layer.*

Penentuan jumlah *neuron* pada *input layer*

* Jumlah neuron sesuai dengan jumlah fitur pada data input

Penentuan jumlah *neuron* pada *output layer*

* Jumlah neuron sesuai dengan permasalahan
* Pada permasalahan klasifikasi biner dan regresi bisa menggunakan hanya satu *neuron*
* Pada permasalahan klasifikasi *multiclass* menggunakan jumlah *neuron* sesuai jumlah label kelasnya, misalnya: 10 neuron pada pengenalan angka

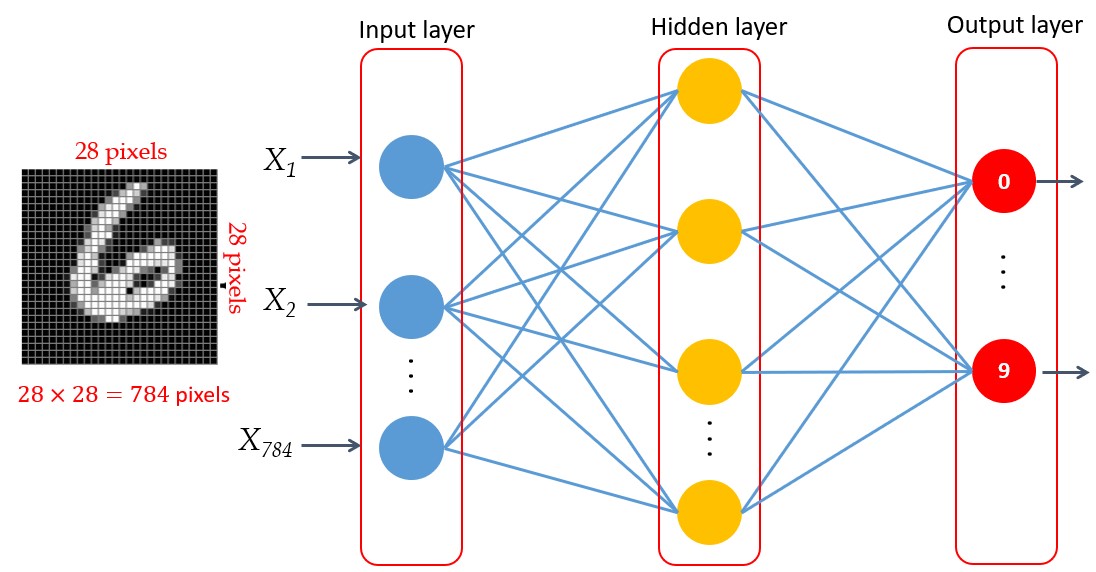
Penentuan jumlah *hidden layer*:

* Semakin banyak jumlah layer memerlukan komputasi waktu lebih lama
* Jumlah layer sebaiknya disesuaikan dengan kompleksitas permasalahan

Penentuan jumlah node (neuron) pada *hidden layer*:

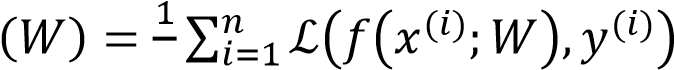
## • Semakin banyak jumlah node memungkinkan mempelajari pola yang lebih rumit • Untuk mencegah overfitting sebaiknya menambah jumlah node secara bertahap

Contoh desain aristektur Multi-layer ANN untuk pengenalan angka dapat dilihat pada Gambar 5.

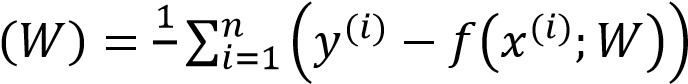


## Gambar 5. Contoh desain arsitektur ANN pada pengenalan angka

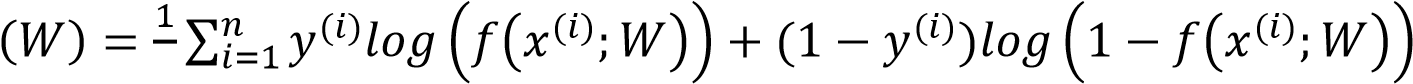
Proses pembelajaran pada metode ANN adalah proses mendapatkan nilai bobot yang optimal pada setiap koneksi antara neuron yang memenuhi kriteria fungsi obyektif. Fungsi obyektif metode ANN berupa fungsi *loss* seperti pada persamaan 6. Persaman 7 merupakan fungsi *Means square loss* dan persamaan 8 merupakan fungsi *Cross entropy loss*. Fungsi *Means square loss* umumnya digunakan pada permasalahan regresi, sedangkan pada permasalahan klasifikasi menggunakan fungsi *Cross entropy loss*. Pada kasus permasalahan klaisifikasi biner, umumnya menggunakan fungsi *Binary Cross entropy loss*.

𝐽 𝑛 (6)

2

𝐽 (7)

𝑛

𝐽 (8)

𝑛

# Algoritma Pembelajaran Backpropagation

Proses pembelajaran pada ANN bertujuan untuk mendapatkan bobot-bobot yang optimal melalui proses meminimalisasi fungsi obyektif. Algoritma *backpropagation* merupakan algoritma pembelajaran yang digunakan oleh metode ANN dalam proses mengupdate nilai bobot. Pembelajaran dengan algoritma *backpropagation* terdiri atas tiga tahap, yaitu (i) *feed forward*, (ii) propagasi *error*, dan (iii) pembaruan bobot dan bias.

Tahap satu adalah *feed forward* yaitu menghitung luaran dari semua neuron yang diaktifkan menggunakan suatu fungsi aktivasi. Pada tahap *feed forward* ini, persamaan 1 digunakan untuk estimasi nilai luaran setiap neuron. Sinyal luaran setiap neuron ini kemudian dikirim ke seluruh neuron pada layer selanjutnya hingga *output layer*.

Tahap kedua adalah menghitung *propagate* *error* antara hasil perhitungan nilai luaran pada output layer (nilai prediksi) dengan nilai target yang sudah ditentukan pada data latih. Tahap ketiga adalah mengupdate nilia bobot pada semua layer dengan fungsi tujuan adalah meminimalkan fungsi loss seperti pada persamaan 9 dan 10.

𝑊∗ = argmin 𝐽(𝑊) (9)

𝑊

𝑊∗ = argmin 1 (10)

𝑊 𝑛

Algoritma pembelajaran pada metode ANN umumnya menggunakan algoritima *backpropagation gradient descent*.

Algoritma *Backpropagation* *Gradient descent*

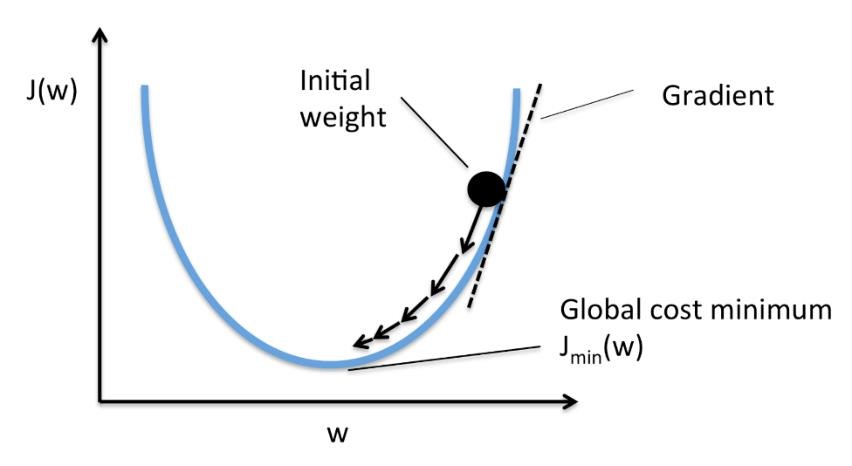
* Inisialisasi bobot secara random
* Iterasi sampai konvergen
* Hitung *gradient,* 𝜕𝐽(𝑊)

𝜕𝑊

* Update bobot, 𝑊 ← 𝑊 − 𝜂 𝜕𝐽(𝑊)

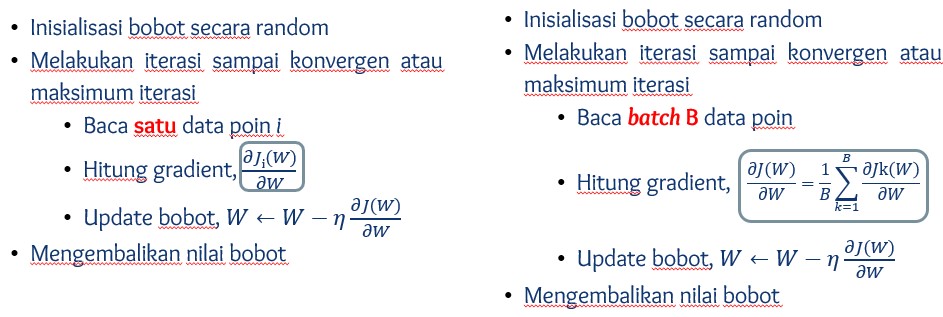
𝜕𝑊

* Mengembalikan nilai bobot



Gambar 6. Grafik optimasi nilai bobot dengan fungsi obyektif

# Algoritma Stochastic Gradient Descent



# Tahapan Pembelajaran Multi-layer Perceptron ANN

Langkah 0 – Inisialisasi bobot, *learning rate*, maksimum iterasi

Langkah 1 – Membaca vektor input *X*

Langkah 2 – Lakukan iterasi (*epoch*)

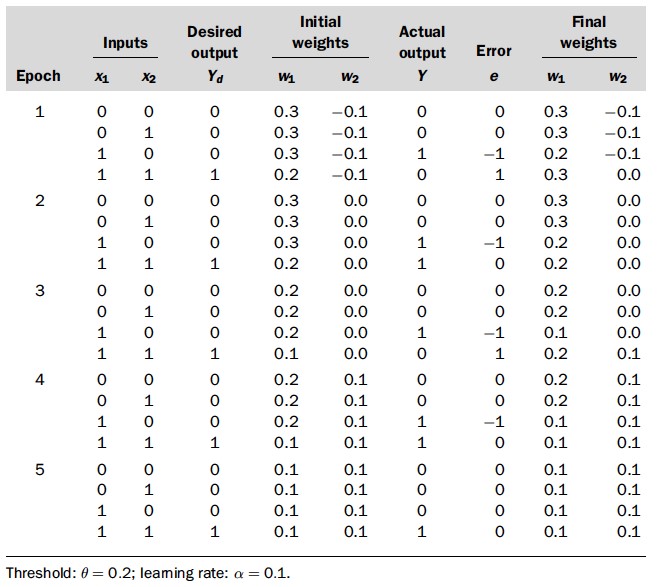
Langkah 3 – Hitung luaran neuron di hidden layer dan output layer

Langkah 4 – Hitung *back propagate error* (pada output layer dan hidden layer)

Langkah 5 – Perbarui semua bobot (pada output layer dan hidden layer)

Langkah 6 – Ulangi langkah 3 – 5 hingga bobot konvergen atau maksimum iterasi Langkah 7 – Luaran berupa matrik bobot (pada output layer dan hidden layer)

Contoh perhitungan dari penerapan algoritma pembelajaran perceptorn Multi-layer ANN pada kasus operasi logika AND dapat dilihat pada tabel dibawah ini:



Sumber: MICHAEL NEGNEVITSKY, Artificial Intelligence: A Guide to Intelligent Systems, Second Edition, Addision Wesley, 2005

# Strategi Pembelajaran Metode ANN

Ada beberapa strategi pembelajaran pada metode ANN untuk menangani beberapa kendala dalam proses pelatihan yaitu:

1. Permasalahan *overfitting*

*Overfitting* adalah permasalahan yang umum dihadapi pada saat membangun model ANN yaitu kinerja dari data latih lebih tinggi daripada kinerja dari data validasi. Strategi yang bisa dilakukan untuk mencegah *overfitting* adalah:

* + regularisasi
  + melakukan augmentasi data
  + penggunaan *dropout* pada saat proses pelatihan
  + Early stopping

1. Permasalahan kinerja model yang rendah

Pada proses pembangunan model terkadang kinerja model masih rendah, sehingga model tersebut belum bisa digunakan untuk proses prediksi pada data uji. Strategi untuk meningkatkan kinerja model yang masih rendah sebagai berikut:

* + Modifikasi *network*
  + Merubah fungsi aktivasi
  + Menggunakan *adaptive learning rate*

# Regularization

Regularisasi dilakukan untuk mengurangi generalization error dengan mencegah model lebih kompleks. Penerapan regularisasi dengan cara menambahkan regularization term pada semua parameter (bobot) ke fungsi obyektif.

* Regularization L1 norm o Menambahkan sum of the absolute weights sebagai penalty term ke fungsi obyektif
* Regularization L2 norm (weight decay) o Menambahkan sum of the squared weights sebagai penalty term ke fungsi obyektif

# Augmentasi Data

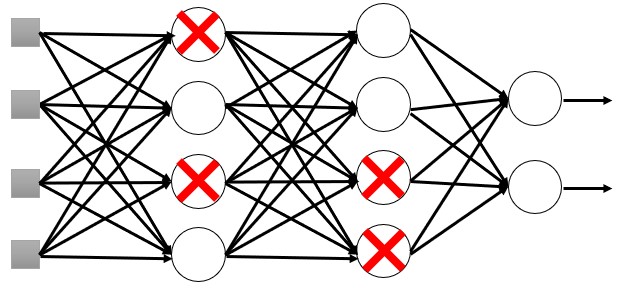
Augmentasi data adalah proses memperbanyak variasi data latih, sehingga model yang dihasilkan lebih baik dalam memprediksi data uji. Metode augmentasi data yang digunakan tergantung dari jenis data input. Pada data citra, metode augmentasi yang umum digunakan adalah rotasi*,* translasi, *flip*, dan *zoom* (dapat dilihat pada Gambar 7).



Gambar 7. Ilustrasi penggunaan Augmentasi Data

# Konsep *Dropout*

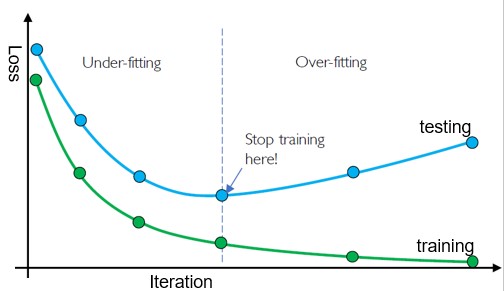
*Dropout* adalah cara untuk meregulasi parameter untuk menghindari *overfitting* sehingga model lebih general. Umumnya dilakukan pada *Fully Connected Layer*. Selama proses training, setiap iterasi sebelum menghitung gradient dilakukan penentuan neuron yang diset tidak aktif sesuai prosentase dropout *p*%. Contoh ilustrasi konsep Dropout pada saat proses pelatihan dapat dilihat pada Gambar 8.



Gambar 8. Ilustrasi penggunaan *Dropout*

## **Konsep *Early Stopping***

Iterasi pada saat training dihentikan jika generalization error mulai naik seperti pada Gambar 9.



Gambar 9. Ilustrasi penggunaan *Early Stopping*

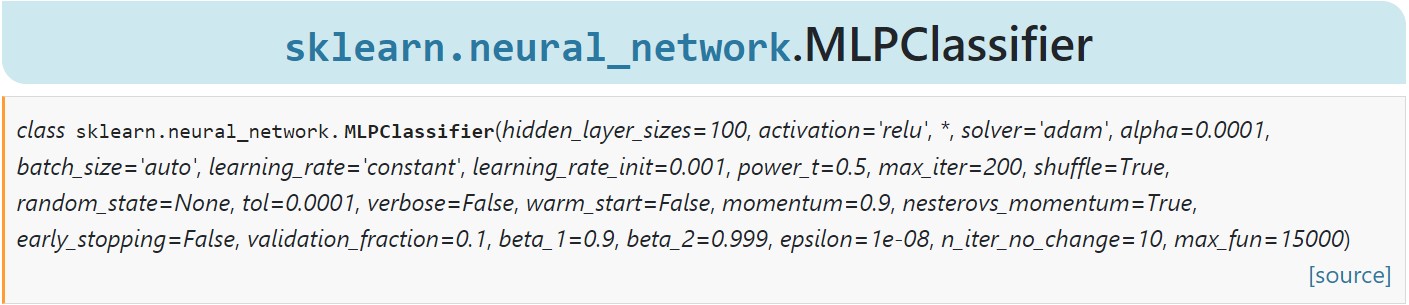
## ***Adaptive Learning Rate***

Nilai *learning rate* berpengaruh pada perhitungan bobot baru, umumnya penggunaan *learning rate* yang menyesuaikan nilai gradien (*adaptive learning rate*) menunjukkan kinerja model yang lebih baik. Contoh *algoritma adaptive learning rate*:

* SGD [Kiefer & Wolfowitz, Ann. Math. Statits. 1952]
* Adagrad [John Duchi, JMLR 2011]
* Adadelta [Matthew D. Zeiler, arXiv 2012]
* AdaSecant [Caglar Gulcehre, arXiv 2014]
* Adam [Diederik P. Kingma, ICLR 2015]
* RMSProp https://www.youtube.com/watch?v=O3sxAc4hxZU

# Tahapan Implementasi ANN

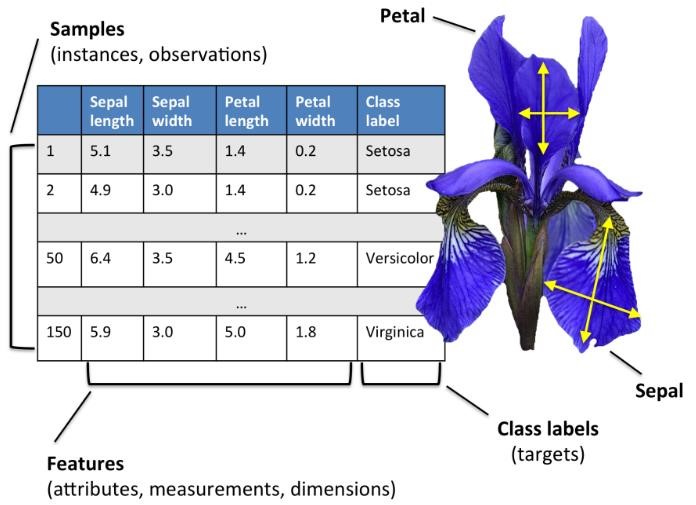
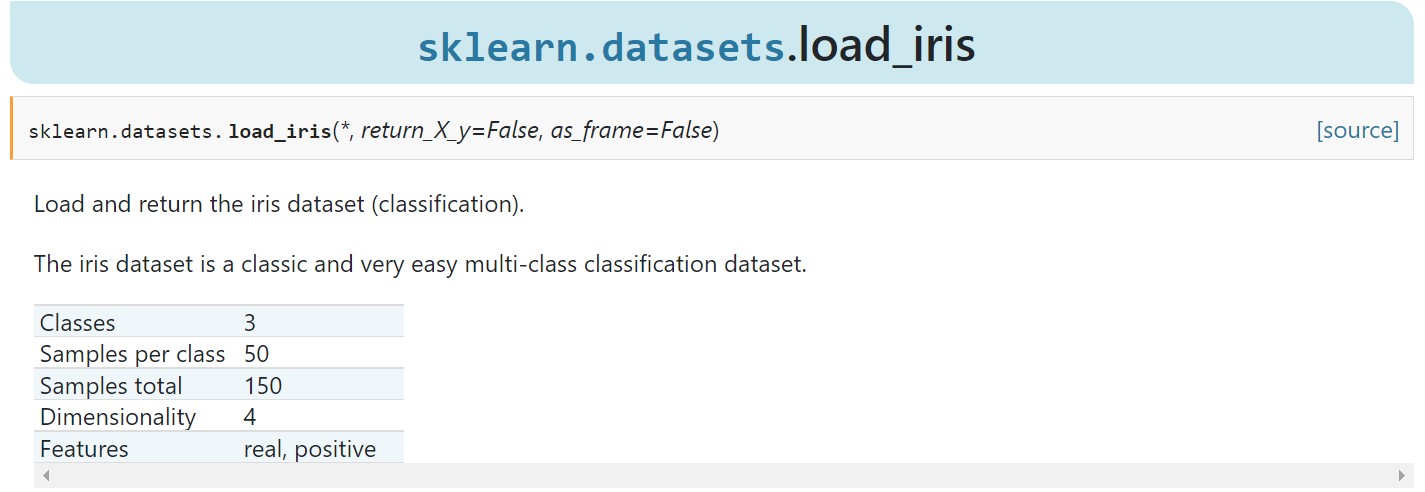
1. Load data: membaca file data input
2. Split data: membagi data menjadi data latih, data validasi, data uji
3. Define model: merancang arsitektur atau model ANN
4. Compile model: menjalankan model ANN yang sudah dirancang
5. Fit model: membangun model ANN berdasarkan data latih
6. Evaluation model: mengevaluasi model ANN berdasarkan data validasi
7. Prediction: memprediksi output dari data uji menggunakan model ANN yang terbaik **TOOLS: Scikit-learn**



Parameter MLPClassifier:

* + hidden\_layer size: jumlah neuron di *hidden layer*
  + activation: fungsi aktivasi yang digunakan di *hidden layer*
  + solver: metode adaptive learning rate yang digunakan
  + batch\_size: ukuran batch
  + learning\_rate\_init: inisialisasi *learning rate*
  + max\_iter: *maksimum iterasi*
  + early\_stopping: bernilai false jika tidak menerapkan *early stopping*

# Contoh implementasi ANN pada dataset Iris menggunakan library Scikit-learn 1. Load Data



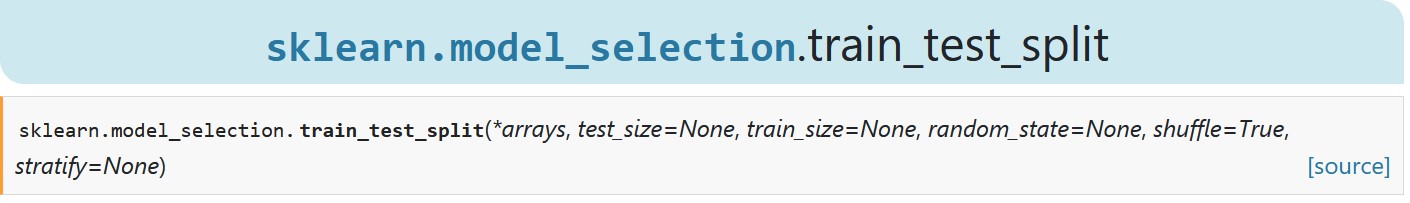
from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data

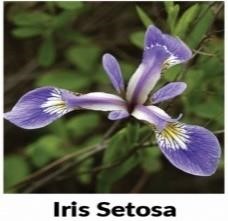
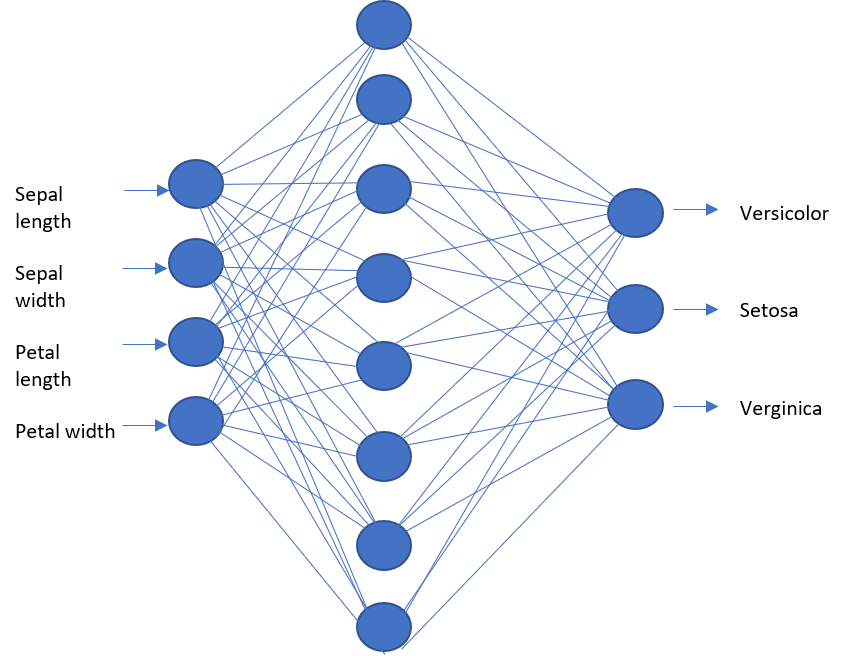
y = iris.target

## **2. Split Data**



|  |
| --- |
| from sklearn.model\_selection import train\_test\_split    X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=.10) X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X\_train, Y\_train, test\_size=.15) print('X\_train', X\_train.shape)    print('X\_val', X\_val.shape) print('X\_test', X\_test.shape) |

## **3. Define Model**



from sklearn.neural\_network import MLPClassifier

mlp = MLPClassifier(**hidden\_layer\_sizes**=(64, ), **activation**='relu’, **learning\_rate\_init**=0.001*,* **max\_iter**=100)

1. **Fit Model dan Evaluation Model**

|  |
| --- |
| from sklearn.metrics import accuracy\_score  mlp.fit(X\_train, Y\_train)  prediksi\_val = mlp.predict(X\_val)  acc\_val = accuracy\_score(Y\_val, prediksi\_val)  print('Akurasi Validasi Training ANN:', acc\_val) |

1. **Prediction**

|  |
| --- |
| from sklearn.metrics import accuracy\_score, plot\_confusion\_matrix  prediksi\_test = mlp.predict(X\_test) acc\_test = accuracy\_score(Y\_test, prediksi\_test) print('Akurasi Testing ANN:', acc\_test) plot\_confusion\_matrix(mlp, X\_test, Y\_test) |

# TOOLS: TENSOR FLOW DAN KERAS

* TensorFlow merupakan *end-to-end open-source platform* untuk mesin pembelajaran ([https://www.tensorflow.org/overview/)](https://www.tensorflow.org/overview/)
* Keras merupakan sebuah deep learning API berbasis Python, berjalan diatas platform TensorFlow. Dimana fokus pengembangan adalah untuk mempercepat proses eksperimen. [(https://keras.io/getting\_started/)](https://keras.io/getting_started/)

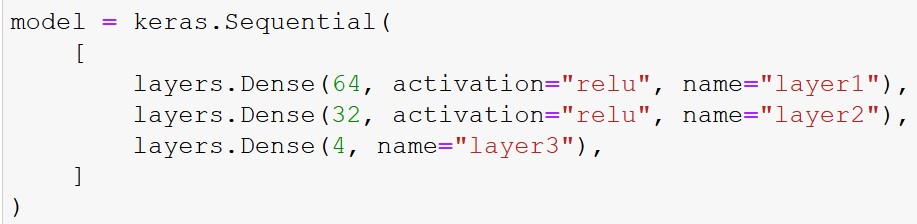


# Keras Model Sequential dan Layers

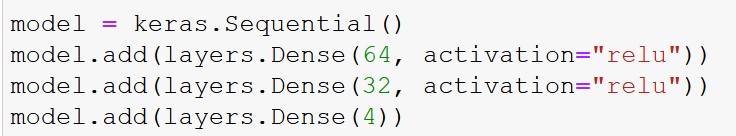
Pembangungan model pada Keras dapat dilihat pada link berikut ini:

<https://keras.io/api/models/>dan <https://keras.io/guides/sequential_model/>

Contoh pembangunan model Sequential pada Keras seperti berikut ini:



atau



# Contoh implementasi ANN pada dataset Iris menggunakan library Keras

## **1. Load Data dan Split Data**

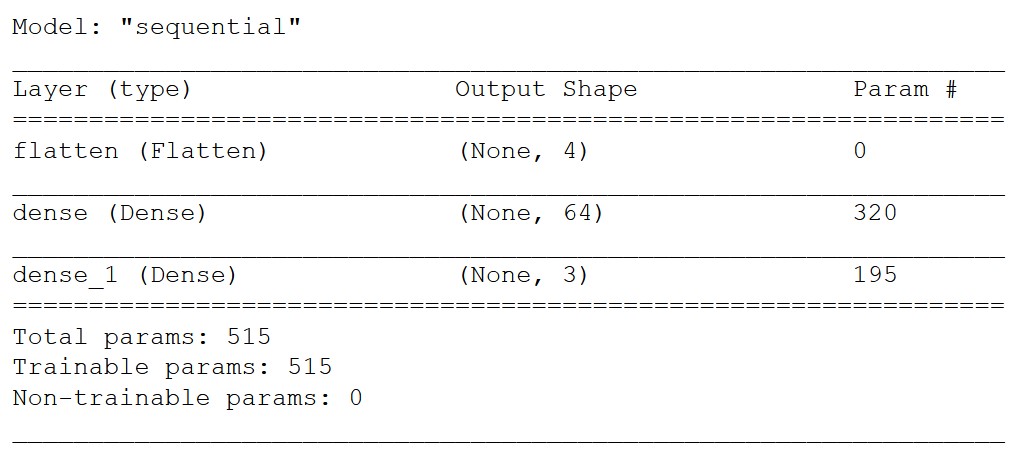
|  |
| --- |
| from sklearn import datasets from sklearn.model\_selection import train\_test\_split  **from keras.utils import to\_categorical**  iris = datasets.load\_iris()  X = iris.data y = iris.target  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=.10)  X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X\_train, Y\_train, test\_size=.2) print('X\_train', X\_train.shape)  print('X\_val', X\_val.shape)  print('X\_test', X\_test.shape)  Y\_train = to\_categorical(Y\_train,3)  Y\_val = to\_categorical(Y\_val,3)  Y\_test = to\_categorical(Y\_test,3) |

## **2. Define Model dan Compile Model**

|  |
| --- |
| from keras.models import Sequential from keras.layers import Flatten, Dense  model = Sequential() model.add(Flatten()) model.add(Dense(64,activation='relu')) model.add(Dense(3,activation='softmax’)) model.compile(optimizer='adam',loss='categorical\_crossentropy', metrics=['acc']) |

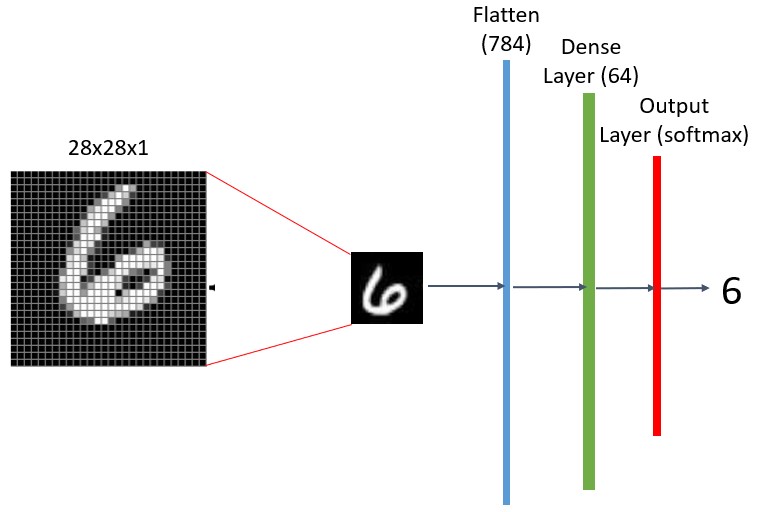
## **3. Fit Model dan Evaluation Model**

|  |
| --- |
| model.fit(X\_train,Y\_train,epochs=64,batch\_size=5,validation\_data=(X\_test,Y\_test)) model.summary()  loss, accuracy = model.evaluate(X\_test, Y\_test) print('Akurasi Testing MLP:', accuracy) |



# Contoh Penerapan Arsitektur ANN Pada Permasalahan Pengenalan Angka

Arsitektur MLP ANN yang akan dibagun untuk pengenalan angka dari citra tulisan tangan seperti pada Gambar 10. Arsitektur MLP yang digunakan adalah *input layer* dengan jumlah neuron sesuai ukuran citra input, jumlah neuron pada *hidden layer* adalah 64, dan jumlah neuron pada *output layer* adalah 10 sesuai jumlah angka yang akan diklasifikasi. Dataset yang digunakan adalah dataset MNITS Handwritten Digit.

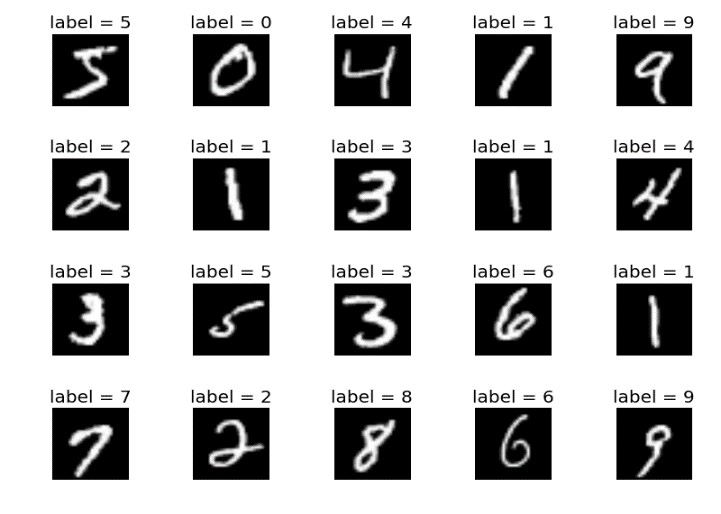


## Gambar 10. Contoh arsitektur Multi-layer ANN untuk pengenalan angka

Dataset MNIST Handwritten Digit dibagi menjadi 3 bagian:

* 55,000 training data
* 10,000 test data
* 5,000 validation data

Setiap citra berukuran 28 × 28 *pixels* dan label kelas berupa *one hot encoded.* Dibawah ini adalah contoh citra pada dataset MNIST beserta contoh label kelasnya.

* 1. [1 0 0 0 0 0 0 0 0 0]
  2. [0 1 0 0 0 0 0 0 0 0]
  3. [0 0 1 0 0 0 0 0 0 0]
  4. [0 0 0 1 0 0 0 0 0 0]
  5. [0 0 0 0 1 0 0 0 0 0]
  6. [0 0 0 0 0 1 0 0 0 0]
  7. [0 0 0 0 0 0 1 0 0 0]

* 1. [0 0 0 0 0 0 0 1 0 0]

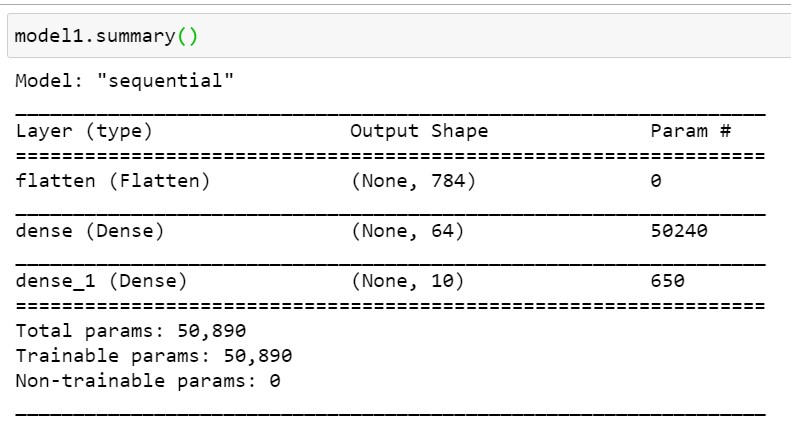
* 1. [0 0 0 0 0 0 0 0 1 0]

### 1. Load Data

|  |
| --- |
| import keras from keras.datasets import mnist    (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()  X\_train = X\_train.reshape(-1, 28,28,1)  X\_test = X\_test.reshape(-1, 28,28,1)  X\_train = X\_train.astype('float32')  X\_test = X\_test.astype('float32')  X\_train /= 255 X\_test /= 255 y\_train = keras.utils.to\_categorical(y\_train, 10) y\_test = keras.utils.to\_categorical(y\_test, 10) |

### 2. Define Model Multi-layer ANN

|  |
| --- |
| from keras.models import Sequential from keras.layers import Flatten, Dense      model1 = Sequential()    model1.add(Flatten())    model1.add(Dense(64,activation='relu'))  model1.add(Dense(10,activation='softmax')) |

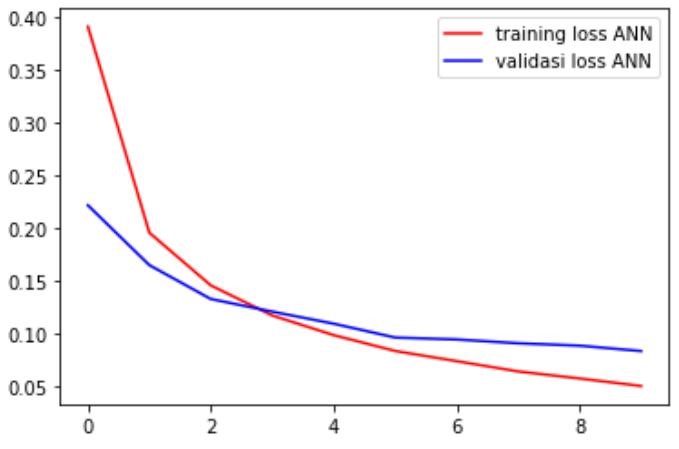


### 3. Model, Fit Model, Save Model, dan Evaluasi Model

model1.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['acc']) history =

model1.fit(X\_train,y\_train,epochs=10,batch\_size=100,validation\_data=(X\_test,y\_test)) model1.save('my\_model1.h5') model1.evaluate(X\_test,y\_test)

# Visualiasasi Evaluasi Model



## **4. Load Model dan Prediction**

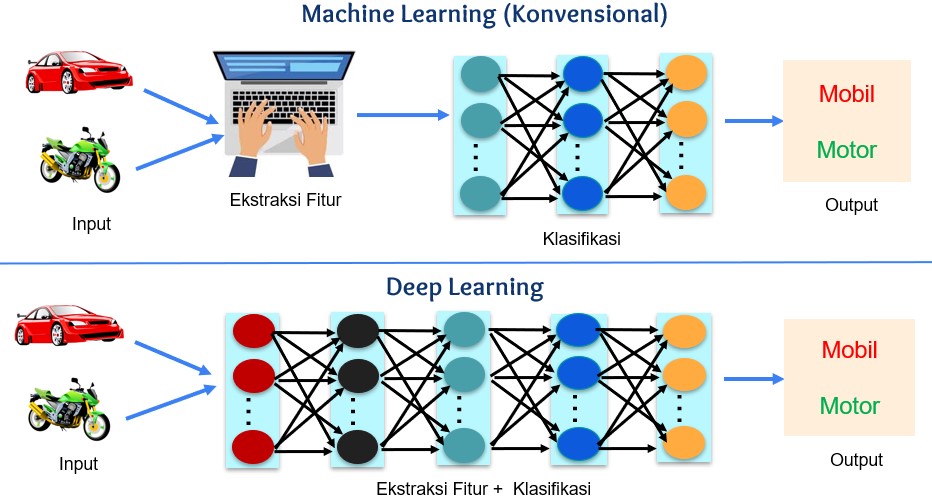
import numpy as np from keras.models import load\_model model\_simpan = load\_model('my\_model.h5') pred = model\_simpan.predict(X\_test) print('label actual:',np.argmax(y\_test[30])) print('label prediction:',np.argmax(pred[30]))

# Pengantar Deep Learning

Pendekatan klasifikasi secara konvensional umumnya melakukan ektraksi fitur secara terpisah kemudian dilanjutkan proses pembelajaran menggunakan metode klasifikasi konvensional (Gambar 11). Kelemahan pendekatan konvensional:

* Memerlukan waktu dan pengetahuan lebih untuk ekstraksi fitur
* Sangat tergantung pada satu domain permasalahan saja sehingga tidak berlaku general

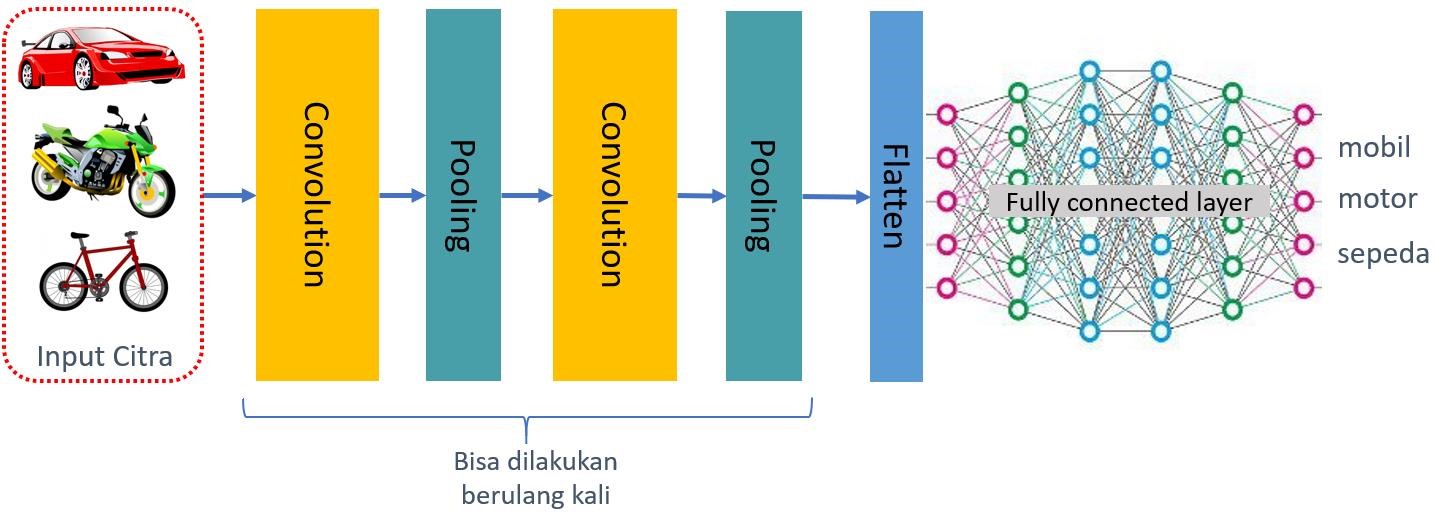
Pendekatan klasifikasi berbasis Deep learning mempelajari representasi hirarki (pola fitur) secara otomatis melalui beberapa tahapan proses *feature learning.*



Gambar 11. Ilustrasi konsep machine learning dan deep learning

# CONVOLUTIONAL NEURAL NETWORK

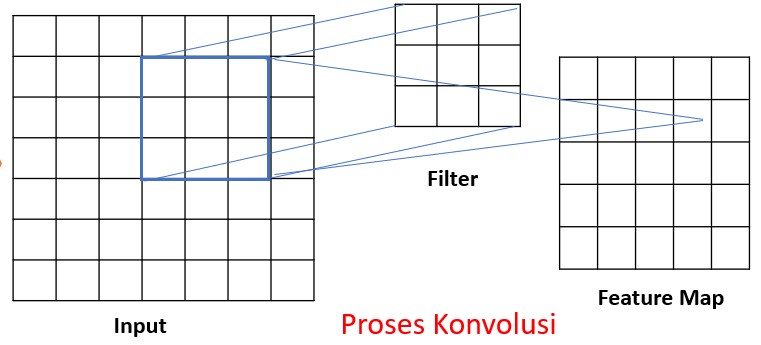
*Convolutional Neural Network* (CNN) adalah salah satu jenis arsitektur dari ANN untuk memproses data yang memiliki susunan topologi seperti grid (I. Goodfellow, dkk., 2016). Misalnya data *time-series* merupakan data yang berbentuk grid 1D, yang menyatakan sampel data terhadap satuan waktu. Contoh lain data citra merupakan data grid 2D yang terdiri dari nilai piksel. Penamaan *Convolutional* karena penggunaan operator konvolusi pada salah satu layer ANN yang menggantikan operasi aljabar matriks biasa. Tahapan pada layer CNN umumnya terbagi menjadi 3 tahap, yaitu: 1) convolution, 2) pooling 3) flatten 4) fully connected layer. Aristektur CNN untuk proses klasifikasi diilustrasikan pada Gambar 12.



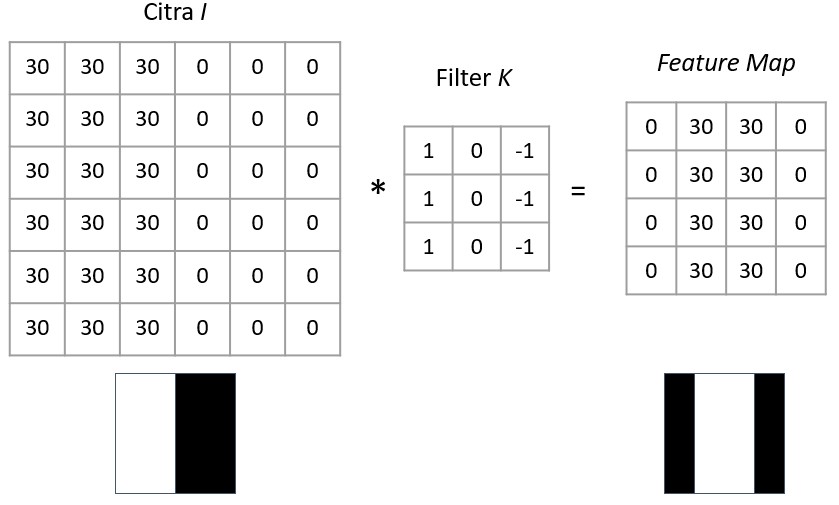
Gambar 12. Arsitektur *Convolutional Neural Network* (CNN)

Tahap pertama merupakan tahap *convolution*, ilustrasi proses konvolusi dapat dilihat pada Gambar 13. Pada tahap ini, melakukan operasi konvolusi pada data input dengan kernel. Kernel pada CNN merupakan *array multidimensional* yang menjadi operator untuk melakukan operasi tertentu dengan array input seperti pada persamaan 11. Nilai yang terdapat pada kernel merupakan nilai yang diperoleh dari hasil learning. Hasil operasi aljabar antara input dengan kernel disebut dengan *feature map*. Proses operasi antara input dengan kernel ini disebut dengan proses ekstraksi fitur ke dalam *feature map*. Contoh proses konvolusi pada sebuah citra dapat dilihat pada Gambar 14.

(𝐼 ∗ 𝐾)(𝑖, 𝑗) == ∑𝑚 ∑𝑛 𝐼(𝑚, 𝑛)𝐾(𝑖 + 𝑚, 𝑗 + 𝑛) (11)

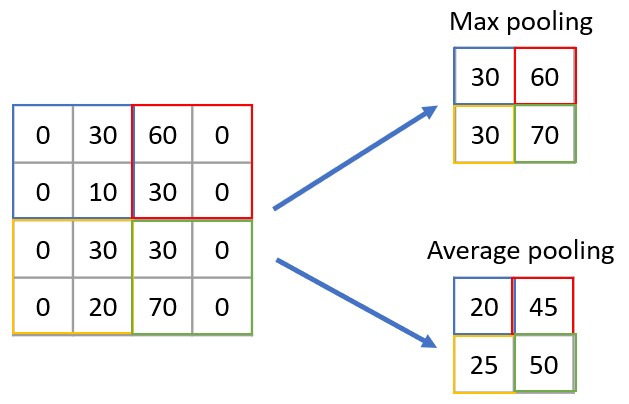


## Gambar 13. Ilustrasi proses Konvolusi



## Gambar 14. Contoh proses Konvolusi pada input citra

Tahap kedua merupakan tahap *detector* yaitu hasil dari tahap sebelumnya dijadikan sebagai input ke dalam suatu fungsi aktivasi misalnya *Rectified linear unit* (ReLU). Tahap ketiga merupakan tahap pooling yaitu mereduksi dimensi dari *input array* yang masuk dengan memanfaatkan fungsi maksimum, penjumlahan, atau rata-rata, yang disebut *maxpooling*, *sum-pooling*, atau *average-pooling.* Luaran dari tahapan ini menghasilkan dimensi data menjadi lebih kecil namun tanpa menghilangkan informasi penting dari data yang diterima. *Pooling layer* yang umum digunakan untuk mengurangi ukuran gambar menjadi lebih kecil (*downsample*) dan mengekstrak *salient features* adalah *Maximum pooling* dan *Average pooling*. Ilustrasi konsep dari *Max-pooling* dan *Averagepooling* dapat dilihat pada Gambar 15.

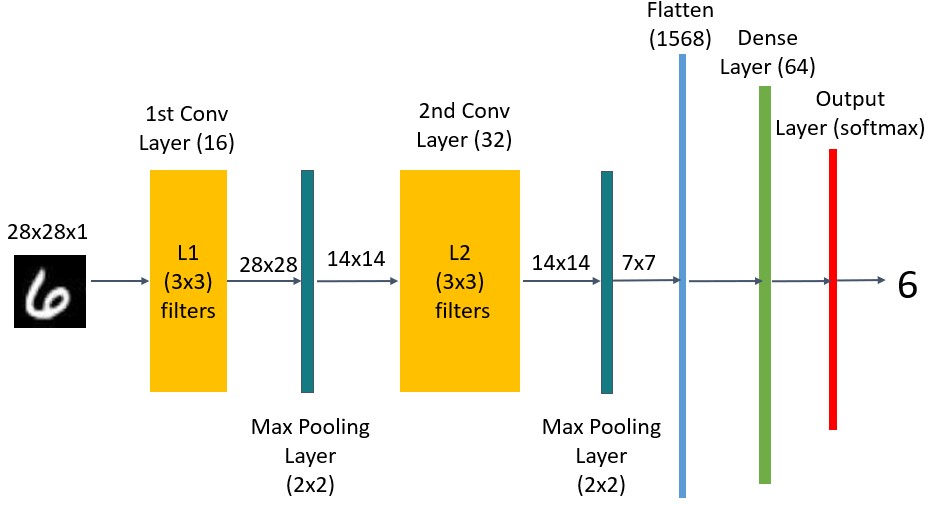


Gambar 15. Contoh proses Pooling pada sebuah *feature map*

Setelah menggunakan beberapa *convolutional* dan *pooling layer*, umumnya untuk permasalahan klasifikasi digunakan layer tambahan berupa *fully connected layer* sebagai tahap 4 yang melakukan proses *flattening* dari matriks menjadi sebuah vektor, dan selanjutnya digunakan untuk melakukan klasifikasi.

# Contoh Penerapan Arsitektur CNN Pada Permaslahan Klasifikasi

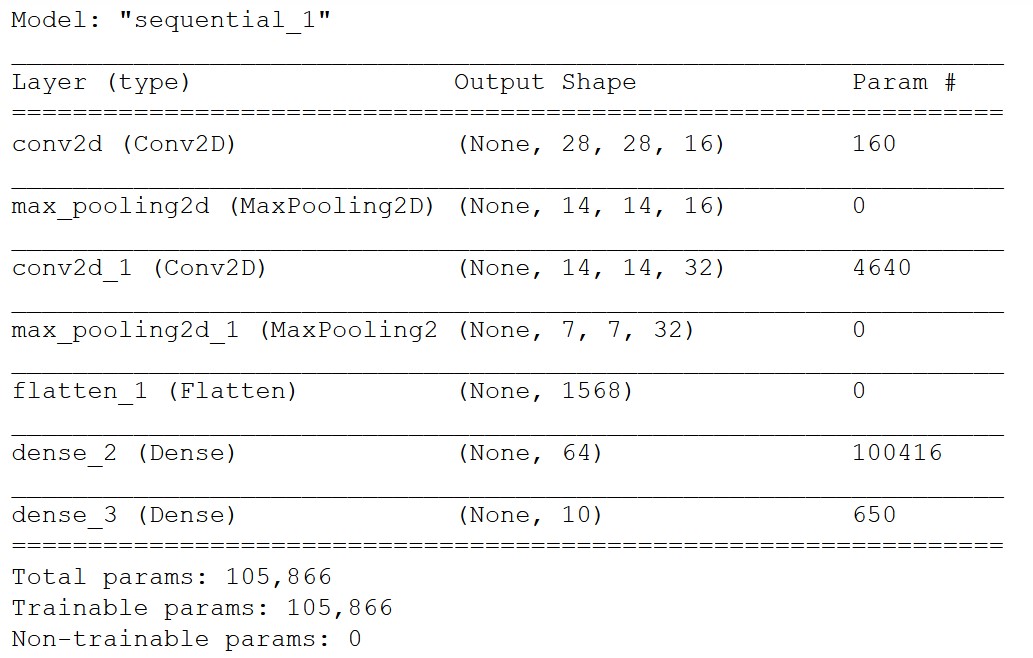
Arsitektur CNN yang akan dibagun untuk pengenalan angka dari citra tulisan tangan seperti pada Gambar 16. Arsitektur CNN yang digunakan adalah *input layer* dengan jumlah neuron sesuai ukuran citra input, dua convolution layer dengan ukuran filter 3x3 sejumlah 16 dan 32, dua max-pooling layer dengan filter 2x2, sedangkan pada arsitektur FCN menggunakan 64 neuron pada hidden layer dan 10 neuron pada *output layer* sesuai jumlah angka yang akan diklasifikasi. Dataset yang digunakan adalah dataset MNITS Handwritten Digit.



Gambar 16. Contoh Arsitektur CNN untuk pengenalan angka

## **1. Define Model, Fit Model dan Evaluasi Model CNN**

|  |
| --- |
| from keras.models import Sequential from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  model2 = Sequential() model2.add(Conv2D(16,(3,3),activation='relu',input\_shape=(28,28,1),padding='s ame')) model2.add(MaxPooling2D(2,2)) model2.add(Conv2D(32,(3,3),activation='relu',padding='same')) model2.add(MaxPooling2D(2,2)) model2.add(Flatten())  model2.add(Dense(64,activation='relu')) model2.add(Dense(10,activation='softmax’)) |



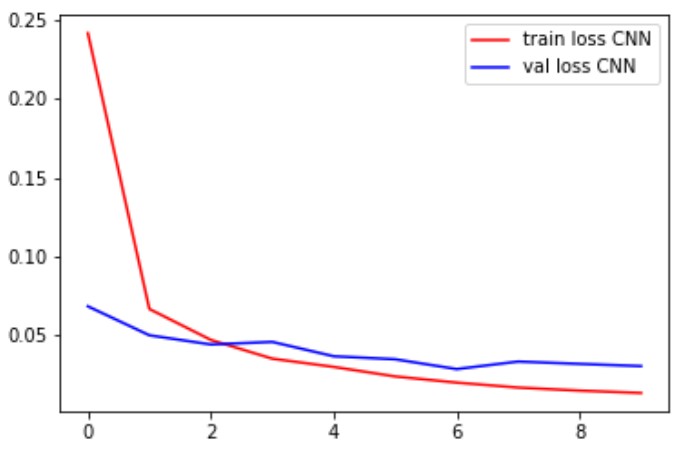
## **2. Compile Model, Fit Model, Save Model, dan Evaluasi Model CNN**

model.

Compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['acc’]) history =

model.fit(X\_train,y\_train,epochs=10,batch\_size=100,validation\_data=(X\_test,y\_ test))

model2.save('my\_model2.h5')



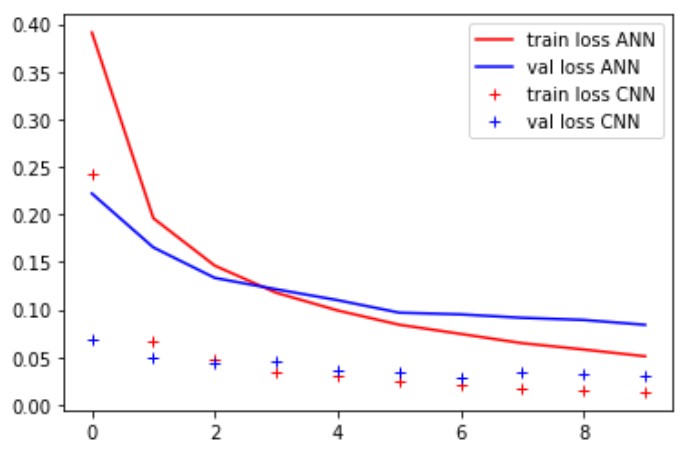
## **3. Load Model dan Prediction**

|  |
| --- |
| import numpy as np from keras.models import load\_model  model\_simpan2 = load\_model('my\_model2.h5') pred = model\_simpan2.predict(X\_test) print('label actual:',np.argmax(y\_test[30])) print('label prediction:',np.argmax(pred[30])) |

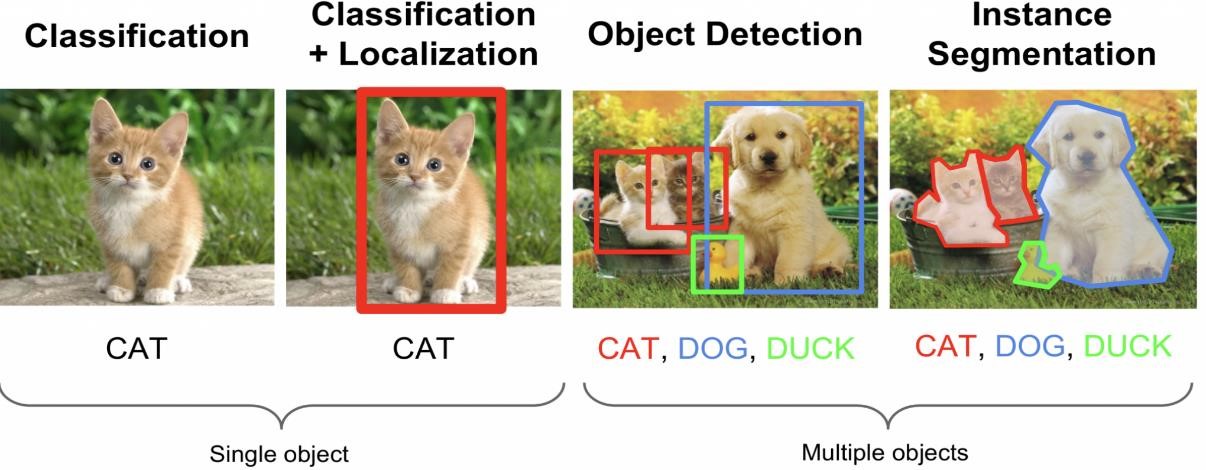
import matplotlib.pyplot as plt

epochs = range(10) loss1 = history1.history['loss'] val\_loss1 = history1.history['val\_loss'] plt.plot(epochs,loss1,'r',label='train loss ANN') plt.plot(epochs,val\_loss1,'b',label='val loss ANN’)

loss2 = history2.history['loss'] val\_loss2 = history2.history['val\_loss'] plt.plot(epochs,loss2,'r+',label='train loss CNN') plt.plot(epochs,val\_loss2,'b+',label='val loss CNN') plt.legend()



Tipe aplikasi yang menggunakan metode CNN misalnya *image classification, object detection*, dan *instance semantic segmentation* seperti pada Gambar 17.



Gambar 17. Contoh tipe aplikasi CNN

# TIPE APLIKASI DAN VARIAN ARSITEKTUR CNN

|  |  |
| --- | --- |
| **Tipe Aplikasi** | **Arsitektur CNN** |
| Image Classification | * LeNet-5 (1998) * AlexNet (2012) * GoogLeNet/Inception (2014) * VGGNet (2014) * ResNet (2015) |
| Object Detection | * R-CNN (2013) * Fast R-CNN (2014) * Faster R-CNN (2015) * Single Shot Detector (SSD) (2016) * YOLO (2016) |
| Semantic (Instance) Segmentation | * Fully Convolutional Network (FCN) (2015) * U-Net (2015) * Feature Pyramid Network (FPN) (2016) * Mask R-CNN (2017 * DeepLab (2016), DeepLabv3 (2017), DeepLabv3+ (2018) |
| Generative model | * Autoencoders, Variational Autoencoders (VAE) * Generative Adversarial Network (GAN) |

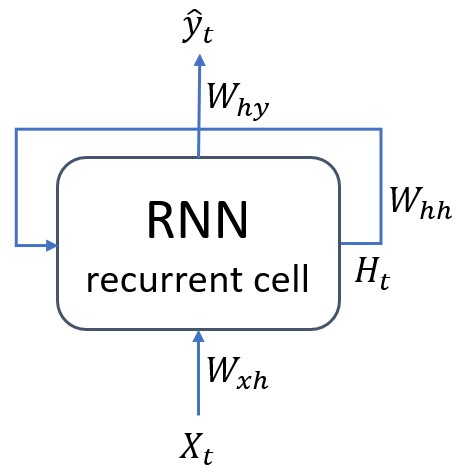
# RECCURENT NEURAL NETWORK

*Recurrent Neural Netword* (RNN) adalah salah satu arsitektur ANN yang mampu merepresentasikan data sequential, misalnya teks, suara, time series, dan sebagainya. RNN berfokus pada sifat data yang mana data sebelumnya mempengaruhi data selanjutnya. Misalnya, diberikan sebuah sequence input x = ( 𝑥1, 𝑥2, 𝑥3.. 𝑥𝑡). Data 𝑥𝑡 dipengaruhi oleh data sebelumnya, ditulis sebagai 𝑃( 𝑥𝑡 | (𝑥1, 𝑥2, 𝑥3.. 𝑥𝑡−1). Ide utamanya adalah memorisasi yaitu mengingat keseluruan sekuens. Sel RNN dapat diilustrasikan pada Gambar 18. Perhitungan hidden state 𝐻 pada waktu ke-*t* bergantung pada fungsi aktivasi 𝜙 dari input pada waktu ke-*t* (𝑋𝑡), hidden state sebelumnya (𝐻𝑡−1) dan parameter pada ANN bias *b*. Konsep ini sesuai dengan prinsip recurrent yaitu mengingat kejadian sebelumnya. Persamaan RNN dapat dilihat pada persamaan 12 dan 13.

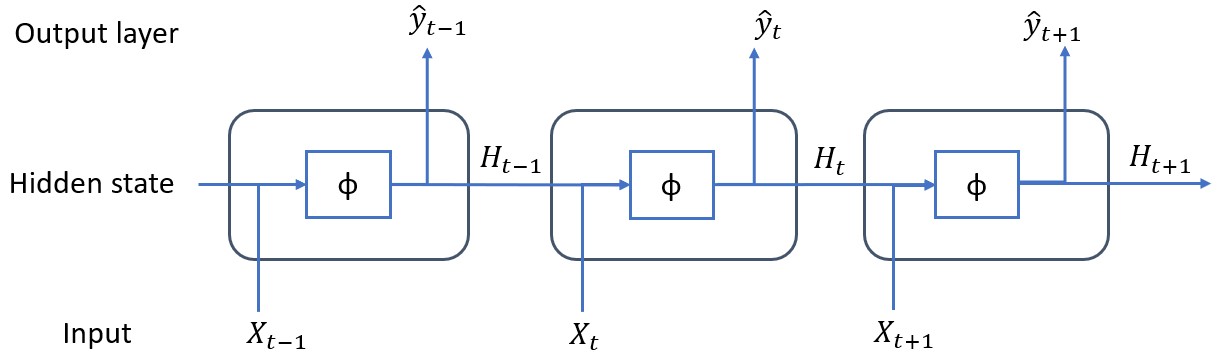
𝑦̂𝑡 = 𝐻𝑡𝑊ℎ𝑦 + 𝑏𝑦 (12)

𝐻𝑡 = 𝜙(𝑋𝑡𝑊𝑥ℎ + 𝐻𝑡−1𝑊ℎℎ + 𝑏ℎ) (13)

Gambar 19 menunjukkan urutan komputasi RNN pada tiga waktu yang berdekatan. Dalam tahap waktu *t*, perhitungan *hidden state* dapat diperlakukan sebagai entri dari lapisan yang terhubung penuh dengan fungsi aktivasi *f* setelah menggabungkan input *Xt* dengan *hidden state* *Ht-1* dari waktu sebelumnya. Output dari lapisan yang terhubung penuh adalah *hidden state* saat ini *Ht*. Parameter modelnya adalah gabungan dari *Wxh* dan *Whh*, dengan bias *bh*. *Hidden state* dari waktu saat ini *t* (*Ht*) akan berpartisipasi dalam menghitung *hidden state* *Ht+1* dari langkah waktu berikutnya *t+1*, yang hasilnya akan menjadi input untuk layer output yang terhubung penuh pada waktu saat ini.



Gambar 18. Arsitektur *Recurrent Neural Network* (RNN)

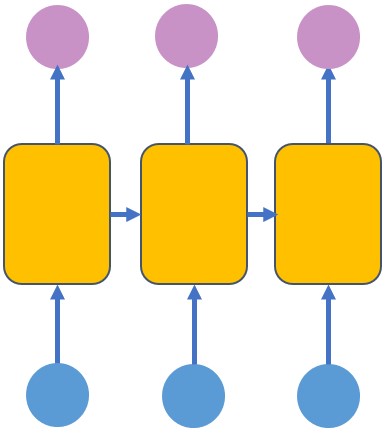
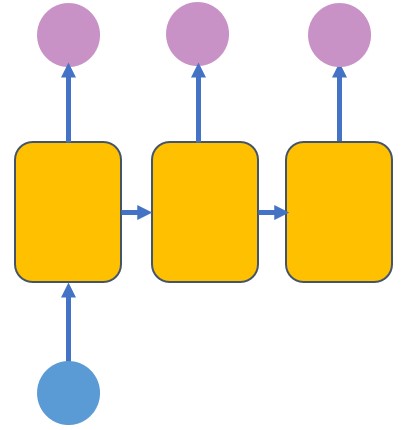
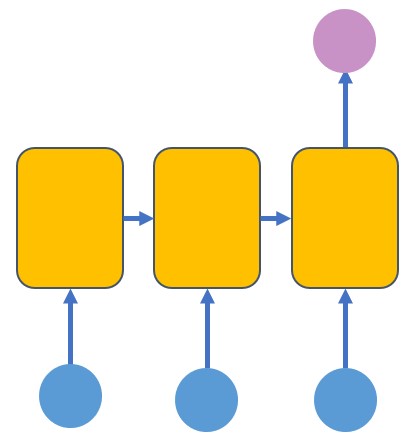


Gambar 19. Arsitektur *Recurrent Neural Network* (RNN)

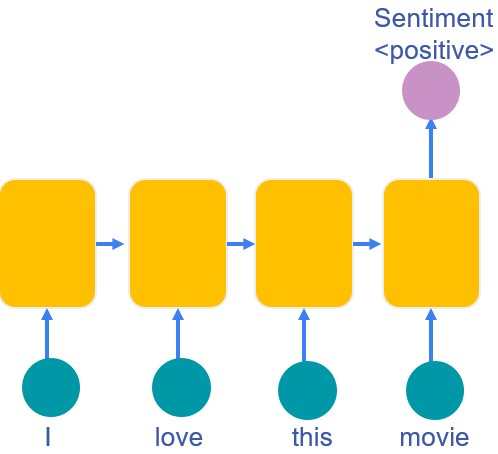
Gambar 20 menunjukkan tipe-tipe arsitektur RNN dan Gambar 21 merupakan contoh penerapan salah satu tipe RNN pada aplikasi analisis sentiment. Contoh-contoh aplikasi dari tipe arsitektur RNN sebagai berikut:

1. *Many to One*: *Sentiment classification, Opinion mining, Speech recognition, Automated answer scoring,* dsb.
2. *One to Many*: *Text generation, Image captioning,* dsb.
3. *Many to Many*: *Translation, Forecasting,* *Music generation, Chatbot,* dsb.

(a) (b) (c)



## Gambar 20. Tipe arsitektur RNN (a) *Many to One* (b) *One to Many* (c) Ma*ny to Many*



Gambar 21. Contoh aplikasi analisis sentiment menggunakan arsitektur RNN

# Varian Arsitektur RNN

Varian dari arsitektur RNN dapat dilihat pada Gambar 22 dan penjelasan setiap arsitektur sebagai berikut:

* Long Short-Term Memory (LSTM)

Merupakan salah satu jenis arsitektur RNN yang terdiri dari beberapa unit yaitu input gate, output gate, dan forget gate. LSTM diusulkan untuk mengatasi permasalahan *vanishing gradient* pada arsitektur RNN.

* Gate Recurrent Unit (GRU)

Merupakan simplifikasi dari arsitektur LSTM dengan menggabungkan input gate dan forget gate sehingga jumlah parameter lebih sedikit.

* Independently RNN (IndRNN)

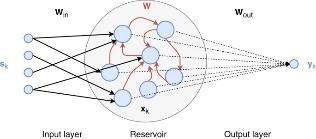
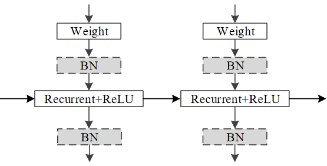
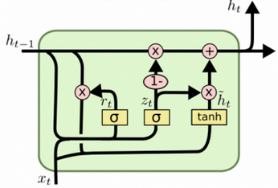
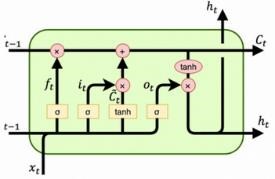
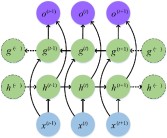
Arsitektur RNN dimana setiap neuron dalam satu layer independen dari yang lain

* Bi-directional RNN

Merupakan arsitektur RNN menghubungkan dua *hidden layer* dari arah yang berlawanan ke output yang sama.

* Echo State Network (ESN)

Ide dasar ESN adalah untuk membuat jaringan berulang yang terhubung secara acak, yang disebut reservoir.



Gambar 22. Varian arsitektur RNN (LSTM, GRU, IndRNN, Bi-directional RNN, ESN)

# REFERENSI

1. 虞台文, Feed-Forward Neural Networks, Course slides presentation
2. Andrew Ng, Machine Learning, Course slides presentation
3. Michael Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, Second Edition, Addision Wesley, 2005.
4. Hung-yi Lee, Deep Learning Tutorial
5. Alexander Amini, Intro to Deep Learning, MIT 6.S191, 2021